Wiki Big Data Project

design document for option 3

Sophia Kholod
Yulianna Tymchenko

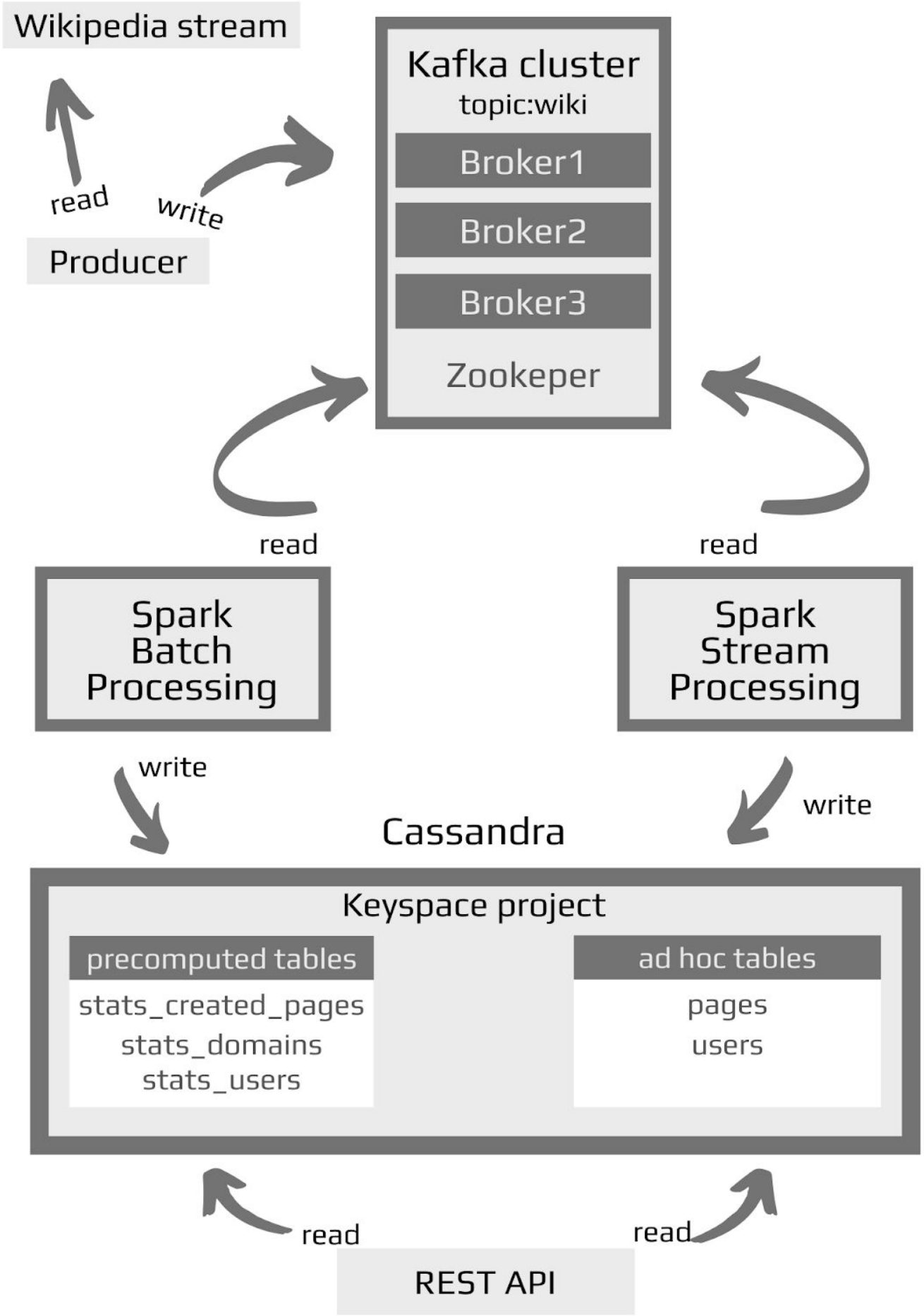
System design

Our task was to parse wiki create pages stream, extract the information and aggregate different statistics.

Example of input data:


```
{
	"$schema":"/mediawiki/revision/create/1.0.0",
	"meta":{
		"uri":"https://www.wikidata.org/wiki/Q96242235",
		"request_id":"0c976b87-451f-4f17-8917-016dc28d3cdd",
		"id":"f1e16e41-6588-4fe8-a8d3-6ab6b0951e8f",
		"dt":"2020-06-12T18:38:43Z",
		"domain":"www.wikidata.org",
		"stream":"mediawiki.page-create",
		"topic":"eqiad.mediawiki.page-create",
		"partition":0,
		"offset":164291608
	},
	"database":"wikidatawiki",
	"page_id":95010239,
	"page_title":"Q96242235",
	"page_namespace":0,
	"rev_id":1205955720,
	"rev_timestamp":"2020-06-12T18:38:43Z",
	"rev_sha1":"q9yeun19pn97zg1vb7tsmzhpmrzituw",
	"rev_minor_edit":false,
	"rev_len":1539,
	"rev_content_model":"wikibase-item",
	"rev_content_format":"application/json",
	"performer":{
		"user_text":"GZWDer (flood)",
		"user_groups":["bot", "*","user","autoconfirmed"],
		"user_is_bot":true,
		"user_id":690508,
		"user_registration_dt":"2014-02-07T14:35:42Z",
		"user_edit_count":20799982
	},
	"page_is_redirect":false,
	"comment":"/* wbeditentity-create-item:0| */ semi-automatic Genealogics import",
	"parsedcomment":"<span dir=\"auto\"><span class=\"autocomment\">wbeditentity-create-item:0|: </span>
semi-automatic Genealogics import</span>"
}
```


Our approach to this problem is the following:


Wikipedia stream
read
write
Producer
Kafka cluster
topic:wiki
Broker1
Broker2
Broker3
Zookeper
read
read
Spark
Batch
Processing
Spark
Stream
Processing
write
write
Cassandra
Keyspace project
precomputed tables
stats_created_pages
stats_domains
stats_users
ad hoc tables
pages
users
read
read
REST API

We decided to use Kafka for storing the pages as makes our system more reliable. Kafka is scalable, fault-tolerant and provides the ability to decouple consumers and producers. And Kafka has a commit log,so that, with Kafka it is unlikely to lose our data.

For the precomputed reports we use the Spark batch processing and send the results to the Cassandra, as it is easily integrated with Spark and if we want to broaden the statistics aggregation to process over than 6 last hours it would be easily achieved. We have to run this processing each hour.

For the ad-hoc queries, we use Spark streaming and store the result in Cassandra again. With this streaming queries we extract the data which is needed for our REST API. We use Cassandra as it is very fast and can store lots of data without the overhead and doesn't need so much computational power.

The reason why we use Spark is that it allows to distribute computation and data, and is fault-tolerant. Moreover, Spark is good for creating different data pipelines and has good integration with both Kafka and Cassandra.

Our systems contains:

3 Cassandra nodes (t2.medium)

3 Spark and Kafka nodes (each node has both Spark and Kafka on it, t2.medium)

Our Application and producer are mainly launched on Spark master node but can be used anywhere with proper configuration.

Thus, we have 6 EC2 instances for this system.

API documentation:

We thought that there is no point in copying it so it can be found in the /docs page -> index.html file.

Results examples:

https://pocbigdata.s3.amazonaws.com/project/users_by_pages.json
https://pocbigdata.s3.amazonaws.com/project/pages_by_bots.json
https://pocbigdata.s3.amazonaws.com/project/pages_by_domains.json

Python implementation:

https://github.com/neverlandjt/www

Chebotko Diagrams for Cassandra:

The copy from the github /docs/cassandra_chebotko_diagrams.pdf diagrams for the system

Keyspace project

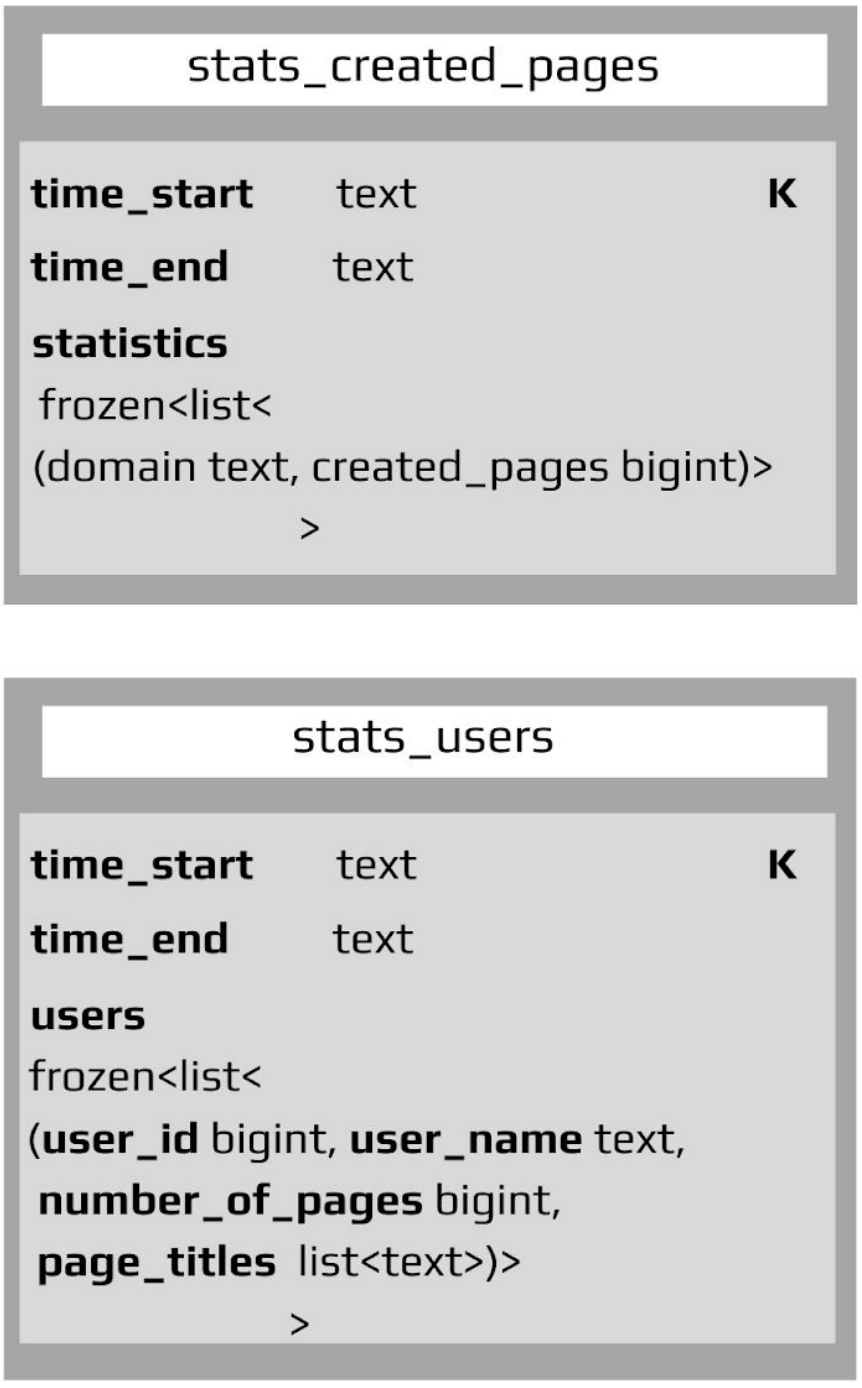
ad hoc queries

users		
id	bigint	**K**
name	text	
page_id	bigint	
timestamp	timestamp	**C↓**

pages		
id	bigint	**K**
url	text	
title	bigint	
domain	text	**C↓**
namespace	int	

precomputed queries

stats_domains		
time_start	text	**K**
time_end	text	
statistics frozen<list< (**domain** text, **created_by_bots** bigint)> >		


stats_created_pages
time_start text K
time_end text
statistics
frozen<list<
(domain text, created_pages bigint)>
>
stats_users
time_start text K
time_end text
users
frozen<list<
(user_id bigint, user_name text,
number_of_pages bigint,
page_titles list<text>)>
>